

Figure 1A

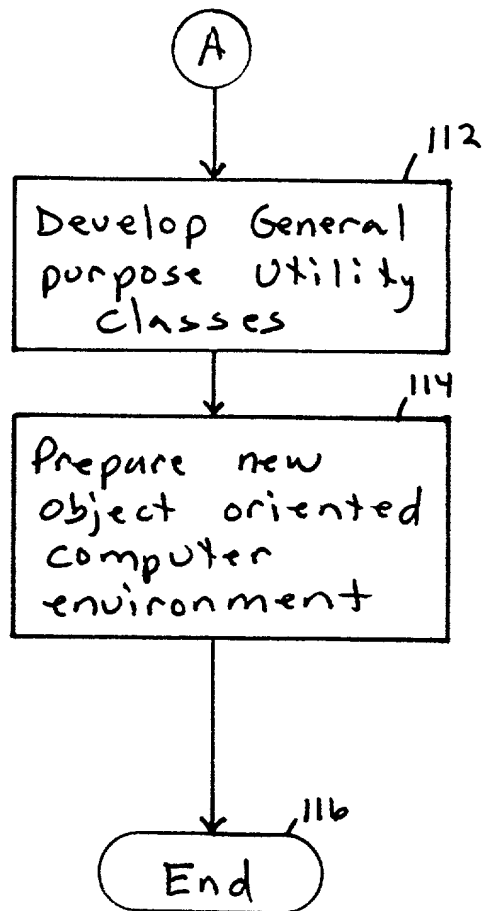


Figure 1B

FIGURE 2

```
// $Header: /2.0/Model/system.adn 27 5/15/98 3:20p Dan $
// System.adn - 05/15/98 09:45

//=====
// Model global controls (intended mainly for regression testing)
// set to 2000 for release 2.0 (the default)
// set to 1300 for regression testing against release 1.3
//=====
Constant DiskAssignmentAlgorithm = 2000;
Constant RandomSeedAssignmentAlgorithm = 2000;

//=====
// Operating system interface constants (must match Strategizer internals)
// *** WARNING: changes in this section will cause execution time failure
//=====

// The Strategizer Operating System Model (a new feature for release 2.0)
//-----

// INTRODUCTION

// The operating system exists as a layer of software logic (and associated processes)
// that lies between software processes running in problem state (as in release 1.3)
// and the underlying hardware.

// An association is made between an operating system name (the first column in the
// CSE.ops file) and an ADN OS behavior name (the sixth column of the CSE.ops file).
// Note that the operating system names are selected from a list (based on the
// CSE.ops file) via the GUI for each computer in a Strategizer model.

// Instances of an operating system are created for each computer that runs an
// operating system with an associated ADN OS behavior name.
// The default for release 2.0 is to define the ADN behavior "ADNOSvc" for all named
// operating systems except "Generic" and "generic_operating_system".
// The operating system name is passed as a second parameter to the ADNOSvc behavior.

// A knowledgeable Strategizer user can create new operating system behaviors
// based on an understanding of the ADNOSvc behavior in this file (System.adn).
// Such user extensions must be added to the end of the System.adn file or included
// via an ADN Include statement at the end of the System.adn file.
```

```

// A few words about the System.adn file. This file is loaded automatically at model
// initialization prior to the processing of ADN source generated or included by the GUI.
// A search is made of the directory containing the model first, then the installation area for
// the System.adn file. The location of the System.adn file selected is logged in the trace file.
// When modifications are planned, it is recommended that a copy of the System.adn file
// be made from the installation area to the directory containing the model.

// ADN PROCESSES AND STATE

// Software processes can execute in problem state and additionally in supervisor
// state (a new feature in release 2.0).

// Processes that startup in problem state switch to supervisor state at specific points
// (OS hook locations) to execute an operating system service and then return to problem
// state.

// Processes that startup in supervisor state (the OS server processes) remain in
// supervisor state.

//-----
// OS behavior hooks are implemented as cases of an ADN switch statement.
// The logic located at each hook is described along side the corresponding
// hook constant.
//-----

Constant INITIALIZEsvc = 0; // The INITIALIZEsvc hook is executed once for each associated
// computer by a special initialization process.
// The purpose of this logic is to establish an operating system instance
// including its server processes and state data.
// Operating system data is maintained uniquely for each OS instance by using
// the functions osSetData and osGetData.
// Any user options and associated processing are included in this section.
// Refer to Case(INITIALIZEsvc) in the ADNosSvc behavior and the associated
// server behaviors ADNosNFS, ADNosVolumeMgr, and ADNosTaskMgr for additional
// information.

// In the following hooks the active process switches from problem to supervisor state and executes the top
// level (or main operating system behavior) in a manner very similar to a behavior call. The hook identifier

```



```
// creates a new task (i.e., process or thread).
// The operating system task count is incremented. If the maximum number
// of tasks has already been reached, the creation of a new task is
// inhibited by blocking the current process (the requestor) until the task
// count drops below the maximum.
// Refer to Case(TASKSTARTsvc) in the ADNosSvc behavior and the ADNosTaskMgr
// behavior for additional details.

Constant TASKENDsvc = 7;
// The TASKENDsvc hook is executed whenever a process or thread terminates.
// The operating system task count is decremented. If the task count is
// greater than the maximum, the first blocked task is allowed to continue.
// Refer to Case(TASKENDsvc) in the ADNosSvc behavior and the ADNosTaskMgr
// behavior for additional details.

//-----
// The "hr" (hardware request data) utility functions are used to access specific data necessary
// to support the current operating system functionality. In release 2.0 this is limited to requests
// generated by the Execute statement.
// The constants defined below are used in combination with the following data access utility
// functions hrGetData/hrSetData to access scalar values, and hrGetDataX/hrSetDataX to access
// vector (or subscripted) values.

// CAUTION: In considering user defined extensions to the operating system the prospective user
// should become familiar with the data currently accessible at the ADN level.
//-----

Constant HRexecSize = 4;
// Used with hrGetData to obtain the size of the Execute request vector.
// The vector consists of the CPU request as first element (when present)
// followed by Read and/or Write requests elements.

Constant HRexecReqType = 5;
// Used with hrGetDataX to obtain the type of the Execute request element.
// Returns one of the following: ReadType, WriteType, SendType, or CpuType.

Constant HRnfsProc = 6;
// Used with hrSetData and hrGetData to save and retrieve the process id
// of the local NFS process.

Constant HRioReq = 7;
// Used with hrGetDataX to obtain the handle to an IO request structure
// (element of the Execute vector).

Constant HRresetReq = 8;
// Used with hrSetDataX to set the specified element in the Execute request
```

FIGURE 2 PAGE - 4 -

```

// vector to null. This action is done when the original request element
// has been replaced by a more detailed operating system representation.

Constant HRlocalIO      = 9; // Used with hrSetData to initiate a local IO request using the specified
// IO request handle.

Constant HRpostExecute  = 10; // Used with hrSetData to post a completion event to the original Execute
// synchronization control logic. (All parts of an Execute statement must
// be completed before a process exits the Execute statement.)

Constant HRkbytes       = 11; // Used with hrGetDataX to obtain the total data bytes (in Kbytes) for the
// specified IO request

Constant HRvolumeHandle = 12; // Used with hrGetDataX to obtain the handle of the associated volume for
// LocVolType and RemVolType io requests.
// The handle is used by volGetStripeSize() and volGetStripedDiskNumber()
// volume manager utility functions.

Constant HRkbytesOffset = 13; // Used with hrGetDataX to obtain the kbyte offset of the first IO record
// in the specified File based on the value of the FirstIo parameter on an
// execute Read or Write request. If FirstIo is not specified a random
// record number between 0 and max-1 is used as the first IO record.
// The offset is used by the volume manager to determine the disk on which
// the first IO record resides.

Constant HRreqType      = 14; // Used with hrGetDataX for Read and Write execute elements.
// Returns one of LocDiskType, RemDiskType, LocVolType, RemVolType.

Constant HRioReqCopy    = 15; // Used with hrGetDataX to make a copy of the specified IO request

Constant HRioReqDisk    = 16; // Used with hrSetDataX to set the disk number for the specified IO request

Constant HRioReqKbytes  = 17; // Used with hrSetDataX to set the ammount of data to be transferred

Constant HRioReqNumber  = 18; // Used with hrSetDataX to set the starting record for the specified IO request

//-----
// Hardware request element type.
// Returned by HRexecReqType when used with hrGetDataX.

```

```
//-----
Constant ReadType      = 0;
Constant WriteType     = 1;
Constant SendType      = 2; // currently not needed
Constant CpuType       = 3;

//-----
// IO request context type.
// Returned by HrReqType when used with HrGetDataX.
//-----
Constant LocDiskType   = 0;
Constant RemDiskType   = 1;
Constant LocVolType    = 2;
Constant RemVolType    = 3;
Constant NonIoType     = 4;

//=====
// ----- end of operating system interface constants -----
//=====

// miscellaneous parameters (used by ADNoNFS behavior)
//-----
Constant RPCreadReq     = 40.0 / 1024.0; // kbytes
Constant RPCwriteAck    = 40.0 / 1024.0; // kbytes

// task manager trace control (use for debugging only)
//-----
Constant TASKtrace     = 0;

//=====
// default operating system service "main" behavior (referenced in CSE.ops)
//=====

Behavior ADNoNfsSvc( svc_type, operating_system_name, computer_name, memory_structure, page_size, instr_per_page ) {
    // NOTE: Only the "svc_type" parameter is available on all but the INITIALIZEsvc case.
}
```

```

Switch( svc_type ) {

    Case( INITIALIZEsvc ) {

        //=====
        // This logic is executed in 0 simulated time to initialize an instance of this
        // operating system on each computer that specifies ADNosSvc in the CSE.ops file.
        // The "operating_system_name" (second behavior parameter) corresponds to the
        // name in column one of the CSE.ops file. This name may be used to differentiate
        // between the initialization of differently named operating systems.
        //=====

        osSetData("svcState",1); // required for initialization process

        //-----
        // Startup Memory Pageout Manager (required by memory model)
        //-----
        Startup proc = MemoryPageoutManager( memory_structure, page_size, instr_per_page )
            Priority 101;
        processSetName(proc,"mpm-"+computer_name);

        // OS service Master controls affect all operating system instances that
        // specify use of the ADNosSvc behavior in column 6 of the CSE.ops file.

        // active tasks control process
        If ( osGetData("taskCountMax") >= 0 ) {
            osSetData("taskCount",0);
            Startup proc = ADNosTaskMgr() Priority 101;
            osSetData("osTaskMgr",proc);
        }

        // NFS server process
        Startup proc = ADNosNFS() Priority 101 Options "SetStatsFlag";
        osSetData("osNFS",proc);
        processSetName(proc,"nfs-"+computerGetName());

        // volume manager
        Startup proc = ADNosVolumeMgr() Priority 101;

```

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

```

osSetData("osVolMgr",proc);
}

Case( EXECUTEsvc ) {

    // this logic is executed in 0 simulated time to send any volume or remote IO requests
    // included in an Execute statement to the local Volume manager or NFS server

    execSize = hrGetData(HRexecSize);
    i = 0;
    While( i < execSize ) {
        reqType = hrGetDataX(HRexecReqType,i);
        Switch ( hrGetDataX(HRreqType,i) ) {
            Case( LocDiskType ) {
                // no OS service required
            }
            Case( RemDiskType ) {
                ioReq = hrGetDataX(HRioReq,i);
                Kbytes = hrGetDataX(HRkbytes,i);
                Send osGetData("osNFS") ("client_side",hrGetDataX(HRnfsProc,i),
                    ioReq,reqType,Kbytes,0,0); // async
                hrSetDataX(HRresetReq,i,0);
            }
            Case( LocVolType ) {
                ioReq = hrGetDataX(HRioReq,i);
                Kbytes = hrGetDataX(HRkbytes,i);
                KbytesOffset = hrGetDataX(HRkbytesOffset,i);
                volumeHandle = hrGetDataX(HRvolumeHandle,i);
                Send osGetData("osVolMgr") (0,ioReq,reqType,Kbytes,KbytesOffset,
                    volumeHandle); // async
                hrSetDataX(HRresetReq,i,0);
            }
            Case( RemVolType ) {
                ioReq = hrGetDataX(HRioReq,i);
                Kbytes = hrGetDataX(HRkbytes,i);
                kbytesOffset = hrGetDataX(HRkbytesOffset,i);
                volumeHandle = hrGetDataX(HRvolumeHandle,i);
                Send osGetData("osNFS") ("client_side",hrGetDataX(HRnfsProc,i),
                    ioReq,reqType,Kbytes,

```

FIGURE 2 PAGE - 8 -

```

        kbytesOffset,volumeHandle); // async
    hrSetDataX(HRresetReq,i,0);
}
Case( NonIoType ) {
    // no OS service required
}
}
i = i + 1;
}

Case( SENDsvc ) {
    // Execute Cpu 0.0000001;
}
Case( SENDWAITsvc ) {
    // Execute Cpu 0.0000001;
}
Case( RECEIVESvc ) {
    // Execute Cpu 0.0000001;
}
Case( REPLYsvc ) {
    // Execute Cpu 0.0000001;
}

Case( TASKSTARTsvc ) {
    // increment task count
    taskCount = osGetData("taskCount") + 1;
    osSetData("taskCount",taskCount);

    // if task count exceeds max put new task in task manager's queue
    // and put new task into wait state
    If ( taskCount > osGetData("taskCountMax") ) {
        Send osGetData("osTaskMgr") (threadGetCurrentId());
        If ( TASKtrace ) {
            Print stringFormat("%6f",simGetTime()),
                "ADNosTaskMgr: task",threadGetCurrentId(), "suspended";
        }
        threadWaitForSignal();
    }
}

```

```

}
Case( TASKENDsvc ) {
    // decrement task count
    taskCount = osGetData("taskCount") - 1;
    osSetData("taskCount", taskCount);

    // if there is a waiting task, signal task manager
    If ( taskCount >= osGetData("taskCountMax") ) {
        processSignal( osGetData("osTaskMgr") );
    }
}
Return( svc_type );
}

// Maximum number of active tasks manager behavior
// -----
Behavior ADNosTaskMgr() {
    While( 1 ) {
        // wait for signal from TASKENDsvc
        processWaitForSignal();

        // remove first task from input queue and signal it
        Receive( task_id ) {
            If ( TASKtrace ) {
                Print stringFormat("%6f", simGetTime()),
                    "ADNosTaskMgr: task", task_id, "resumed";
            }
            threadSignal( task_id );
        } Reply();
    }
}

// NFS server behavior
// -----
Behavior ADNosNFS() {
    osSetData("svcState", 1);
    processSetNoThreadUtilizationStats();
}

```

FIGURE 2 PAGE - 10 -

```

While( 1 ) {
    Receive(type,arg1,arg2,arg3,arg4,arg5,arg6) Thread {
        Switch( type ) {
            Case( "client_side" ) {
                // save client process id
                execute_proc = messageGetSendingProcessId();
                processSetClientProcessId( execute_proc ); // c_proc->client_proc_sn = execute_proc
                If ( arg3 == ReadType ) {
                    msgSendLength = RPCreadReq;
                    msgReplyLength = arg4;
                }
                Else { // WriteType
                    msgSendLength = arg4;
                    msgReplyLength = RPCwriteAck;
                }
            }
            // forward request to remote server
            // ( msg->client_proc_sn = c_proc->client_proc_sn )
            Send arg1("server_side",arg2,arg3,arg4,arg5,arg6,execute_proc) Message msgSendLength
                Protocol "UDP/IP" Wait();

            // post completion event to Execute statement synchronization control
            If ( ! arg6 ) { // not a volume manager request
                hrSetData(HR-postExecute,execute_proc);
            }
        }
        Case( "server_side" ) {
            processSetClientProcessId( arg6 ); // execute_proc
            If ( arg5 ) {
                // volume request
                Send osGetData("osVolMgr") (arg6,arg1,arg2,arg3,arg4,arg5); // async
            }
            Else {
                // disk request
                If ( arg2 == ReadType ) {
                    msgReplyLength = arg3;
                }
                Else { // WriteType
                    msgReplyLength = RPCwriteAck;
                }
            }
        }
    }
}

```

```

        }
        } Reply() Message msgReplyLength;
    }

    Behavior ADNOSVolumeMgr() {
        osSetData("svcState",1);
        processSetNoThreadUtilizationStats();
        While( 1 ) {
            Receive( execute_proc, io_req, req_type, req_kbytes, first_kbytes_offset, volume_handle ) Thread {

                If ( !execute_proc ) { // local request
                    execute_proc = messageGetSendingProcessId();
                }
                processSetClientProcessId( execute_proc );

                // collect statistics
                volBeginRequest( volume_handle, execute_proc );
                request_start_time = simGetTime();

                // for each volume IO request in Execute statement
                kbytes_offset = first_kbytes_offset;
                kbytes = req_kbytes; // total bytes in this I/O request ( Bytes * Number )
                stripe_kbytes = volGetStripeSize(volume_handle);

                // process first stripe, partial stripe up to a stripe boundary, or full request
                mod_kbytes_offset = RMod(kbytes_offset,stripe_kbytes);
                curr_kbytes = RMin(stripe_kbytes-mod_kbytes_offset,stripe_kbytes);

                // modify Number field of original request (to avoid setting it each time)
                hrSetDataX(HRioReqNumber,io_req,1);

                // save the disk number as the reference point for a complete pass through
                // all of the disks in the volume
                first_disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);
            }
        }
    }
}

```

FIGURE 2 PAGE - 12 -

```

Join {

    // loop until all of the data has been processed

    While ( kbytes > 0.0005 ) {

        Join {

            disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);

            // loop over each disk on volume once while there is more data

            While ( ( disk_number >= 0 ) && ( kbytes > 0.0005 ) ) {

                // for each piece of an I/O request
                Thread {

                    // declare client process for associating statistics
                    processSetClientProcessId( execute_proc );

                    If ( kbytes > curr_kbytes ) {

                        // copy original I/O request
                        ioReq = hrGetDataX(HRioReqCopy,io_req);

                    }

                    Else {

                        // use original I/O request
                        ioReq = io_req;

                    }

                    // modify selected fields
                    hrSetDataX(HRioReqDisk,ioReq,disk_number);
                    hrSetDataX(HRioReqKbytes,ioReq,curr_kbytes);

                    // issue local IO request
                    hrSetData(HRlocalIO,ioReq);

                }

                kbytes_offset = kbytes_offset + curr_kbytes;
                kbytes = kbytes - curr_kbytes;
                curr_kbytes = RMin(stripe_kbytes,kbytes);

                disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);
            }

        }

    }

}

```

FIGURE 2 PAGE - 13 -

```

// post completion event to Execute statement synchronization control
// when all pieces of this request have been completed
hrSetData(HR-postExecute,execute_proc);

// collect statistics
volEndRequest( volume_handle, execute_proc, request_start_time );

    } ReplyO;
    }

// user defined OS behavior include statements
//-----
// Include "user_OS_behaviors.adn"; // <== sample syntax

```

FIGURE 3

```
// $Header: /ST/Trunk/Model/system.adn 81 12/19/00 1:11p Dan $
// System.adn - 12/21/2000 09:00
```

```
// -----
// Copyright Hyperformix, Inc., 1996-2000.
// This software, including the program, help files and documentation, is
// owned by Hyperformix, Inc.
// The software contains information which is confidential and proprietary
// to Hyperformix, Inc. Access to and use of the software is available only
// through a nonexclusive license agreement with Hyperformix, Inc.
// The use of this software is controlled by that license agreement and any
// other use or copying of the software will violate the license and is
// expressly prohibited.
// -----
```

312 --package "OperatingSystemPackage";

```
// =====
// Operating system interface constants (must match Strategizer internals)
3202 // *** WARNING: changes in this section will cause execution time failure
// =====
```

```
// The Strategizer Operating System Model has been modified for release 2.2
// to take advantage of the new ADN object-oriented extensions.
// Strategizer users can extend this SES supplied capability by using the
// the new user_extensions.adn option.
// -----
```

// INTRODUCTION

```
// The operating system exists as an instance of the class ses_OperatingSystem
// and a layer of software logic implemented in its behavior methods and
// associated server processes. This layer of logic lies between software
// processes running in application problem state and the underlying hardware.

// An association is made between an operating system name (the first column in the
// CSE.ops file) and an ADN OS behavior name (the sixth column of the CSE.ops file).
// Note that the operating system names are selected from a list (based on the
// CSE.ops file) via the GUI for each computer in a Strategizer model.
```

```

// Instances of an operating system are created for each computer that runs an
// operating system with an associated ADN OS behavior name by invoking that
// behavior to instantiate an OperatingSystem object and call its
// initializeSvc behavior.
// The default for this release is to define the ADN behavior "ADNOSvc" for
// all the named operating systems. The operating system name is passed as a
// parameter to the operating system instance constructor.

// A knowledgeable Strategizer user can create a new operating system class by
// extending the OperatingSystem class supplied by SES in this file (System.adn).
// Such user extensions must be placed in the specially named user_extensions.adn
// file for proper processing.

// ADN PROCESSES AND STATE

// Software processes can execute in problem state and additionally in supervisor
// state (a new feature since release 2.0).

// Processes that startup in problem state switch to supervisor state at specific points
// (OS hook locations) to execute an operating system service and then return to problem
// state.

// Processes that startup in supervisor state (the OS server processes) remain in
// supervisor state.

//-----
// OS behavior hooks are implemented as methods of an instance of the ses_OperatingSystem class
// or a user extension thereof specified in the user_extensions.adn file.
// The logic located at each hook is described along side the corresponding hook constant.
// Note: The hook constant is required on the return from each method as part of the hook
// protocol mechanism.
//-----

Constant INITIALIZEsvc = 0;
// The initializeSvc behavior is executed once for each associated
// computer by a special initialization process after a new instance
// of the ses_OperatingSystem class is created. These actions are taken
// by the ADN OS behavior (named in col. 6 of the CSE.ops file).
// The purpose of this logic is to create the associated server

```

302

```

// processes that make up part of the operating system.
// The operating system state data is maintained in the OperatingSystem
// instance field variables.
// The initializeSvc behavior of the ses_OperatingSystem class should
// be called as the first statement in any initializeSvc overriding
// behavior specified by the user to assure that the basic operating
// system services are properly initialized.
// Refer to the initializeSvc behavior logic for additional details.

// In the following hooks the active process switches from problem to supervisor state and executes the
// corresponding operating system service behavior. The hook constant value is passed back as the only
// return parameter. When the service is completed, the active process returns to problem state.

Constant EXECUTEsvc = 1; // The executeSvc behavior receives control when the Execute statement is ready
// to be sent to the hardware.
// Individual elements in the request vector (prepared from the Execute
// statement) are checked for remote disk IO and IO operations involving
// files located on volumes. Substitution or modification of the original
// requests are made as appropriate. The requests are then passed on to
// the hardware model.
// Refer to the executeSvc behavior logic for additional details.
// Note: It is strongly recommended that this behavior not be overridden
// by the user unless all the original logic is also included.

// The following set of four hooks are designed to work together to provide support for the implementation
// of communication protocol logic. This is expected to be the main part of the operating system logic that
// most users may be interested in extending.
// The service behaviors provided with release 2.2 contain no logic other than to surface addressability
// to the ses_Message object instance associated with the operation. The declaration for the ses_Message class
// is located in the Utilites.adn file.

// The following notes may help in use of the communication service hooks:
// - Synchronous messages execute the following sequence: sendSvc, receiveSvc, replySvc, and sendWaitSvc.
// - Asynchronous messages execute the following sequence: sendSvc then receiveSvc.
// - The sendSvc and replySvc are invoked just before passing control to the hardware.
// - The receiveSvc and sendWaitSvc are invoked just after returning from the hardware.

Constant SENDsvc = 2; // The sendSvc behavior is executed at the end of the software part
// of Send statement processing, just before the request is sent to the
// hardware. Upon exit from this section, the resulting request is sent

```

```

// to the hardware.

Constant SENDWAITsvc = 3;
// The sendWaitSvc behavior is executed early in the processing of an incoming
// message sent by the Reply clause of a Receive statement.
// Upon exit from this section, control is passed to the Wait clause of the
// original Send statement for processing of the message data fields.

Constant RECEIVEsvc = 4;
// The receiveSvc behavior is executed early in the processing of an incoming
// message from a Send statement.
// Upon exit from this section, control is passed to the Receive statement
// for processing of the message data fields.

Constant REPLYsvc = 5;
// The replySvc behavior is executed at the end of the software part
// of the Reply clause (part of the Receive statement), just before the
// request is sent to the hardware.

Constant TASKSTARTsvc
    = 6; // Updates active task count stats
    // Increments active task count
    // Issues warning first time maximum count is issued

Constant TASKENDsvc = 7; // Updates active task count stats
    // Decrements active task count

//-----
// The "hr" (hardware request data) utility functions are used to access specific data necessary
// to support the current operating system functionality. In release 2.0 this is limited to requests
// generated by the Execute statement.
// The constants defined below are used in combination with the following data access utility
// functions hrGetData/hrSetData to access scalar values, and hrGetDataX/hrSetDataX to access
// vector (or subscripted) values.

// CAUTION: In considering user defined extensions to the operating system the prospective user
// should become familiar with the data currently accessible at the ADN level.
//-----

Constant HRexecSize = 4; // Used with hrGetData to obtain the size of the Execute request vector.
    // The vector consists of the CPU request as first element (when present)
    // followed by Read and/or Write requests elements.

Constant HRexecReqType = 5; // Used with hrGetDataX to obtain the type of the Execute request element.

```

FIGURE 3 PAGE - 4 -

Constant HRnfsProc	= 6;	// Returns one of the following: ReadType, WriteType, SendType, or CpuType. // Used with hrSetData and hrGetData to save and retrieve the process id // of the local NFS process.
Constant HRioReq	= 7;	// Used with hrGetDataX to obtain the handle to an IO request structure // (element of the Execute vector).
Constant HRresetReq	= 8;	// Used with hrSetDataX to set the specified element in the Execute request // vector to null. This action is done when the original request element // has been replaced by a more detailed operating system representation.
Constant HRlocalIO	= 9;	// Used with hrSetData to initiate a local IO request using the specified // IO request handle.
Constant HRpostExecute	= 10;	// Used with hrSetData to post a completion event to the original Execute // synchronization control logic. (All parts of an Execute statement must // be completed before a process exits the Execute statement.)
Constant HRkbytes	= 11;	// Used with hrGetDataX to obtain the total data bytes (in Kbytes) for the // specified IO request
Constant HRvolumeHandle	= 12;	// Used with hrGetDataX to obtain the handle of the associated volume for // LocVolType and RemVolType io requests. // The handle is used by volGetStripeSize() and volGetStripedDiskNumber() // volume manager utility functions.
Constant HRkbytesOffset	= 13;	// Used with hrGetDataX to obtain the kbyte offset of the first IO record // in the specified File based on the value of the FirstIo parameter on an // execute Read or Write request. If FirstIo is not specified a random // record number between 0 and max-1 is used as the first IO record. // The offset is used by the volume manager to determine the disk on which // the first IO record resides.
Constant HRreqType	= 14;	// Used with hrGetDataX for Read and Write execute elements. // Returns one of LocDiskType, RemDiskType, LocVolType, RemVolType.
Constant HRioReqCopy	= 15;	// Used with hrGetDataX to make a copy of the specified IO request
Constant HRioReqDisk	= 16;	// Used with hrSetDataX to set the disk number for the specified IO request

FIGURE 3 PAGE - 5 -

```

Constant HRioReqKbytes = 17; // Used with hrSetDataX to set the amount of data to be transferred

Constant HRioReqNumber = 18; // Used with hrSetDataX to set the starting record for the specified IO request

Constant HRlocalVIO = 19; // Used with hrSetData to initiate a local volume manager IO request using the specified
// IO request handle and applying physical attribute.

//-----
// Hardware request element type.
// Returned by HRexecReqType when used with hrGetDataX.
//-----
Constant ReadType = 0;
Constant WriteType = 1;
Constant SendType = 2; // currently not needed
Constant CpuType = 3;

//-----
// IO request context type.
// Returned by HRreqType when used with hrGetDataX.
//-----
Constant LocDiskType = 0;
Constant RemDiskType = 1;
Constant LocVolType = 2;
Constant RemVolType = 3;
Constant NonIoType = 4;

//=====
// ----- end of operating system interface constants -----
//=====

// miscellaneous parameters (used by ADNOSNFS behavior)
//-----
Constant RPCreadReq = 40.0 / 1024.0; // kbytes
Constant RPCwriteAck = 40.0 / 1024.0; // kbytes

```

FIGURE 3 PAGE - 6 -

```

// task manager trace control (use for debugging only)
//-----
Constant TASKTrace    = 0;

//-----
// remote IO distribution policy - used by NFS servers
//-----
public associative gRemoteIoDistributionPolicy[100];

public function registerRemoteIoDistributionPolicy( tComputerName, userRemoteIoDistributionPolicyName ) {
    gRemoteIoDistributionPolicy[tComputerName] = userRemoteIoDistributionPolicyName;
}

//=====
// default operating system service "main" behavior (referenced in CSE.ops)
//=====

public class ses_OperatingSystem {
    314 static integer    fTaskMaxWarningIssued = false;
    static associative fActiveTaskCountStatsPtr[100];
    316
    string            fOpSysName;
    string            fComputerName;
    integer            fMemoryStruct;
    real              fPageSize;
    real              fInstrPerPage;

    integer            fOsMemMgr;
    string            fRemoteIoDistributionPolicy;

    integer            fOsNFS;

    integer            fOsTaskMgr;
    integer            fTaskCountMax = -1;
    integer            fTaskMaxReached = false;
    ses_Statistic      fActiveTaskCountStats = null;
    integer            fActiveTaskCount = 0;

    integer            fOsVolMgr;
}
    306
    310
    304

```

```

ses_ThreadList  fThreadList;

constructor ses_OperatingSystem(aOpSysName,aComputerName,aMemoryStruct,
    aPageSize,aInstrPerPage) {
    fOpSysName = aOpSysName;
    fComputerName = aComputerName;
    fMemoryStruct = aMemoryStruct;
    fPageSize = aPageSize;
    fInstrPerPage = aInstrPerPage;
    fRemoteIoDistributionPolicy = gRemoteIoDistributionPolicy[stringNameBase(aComputerName)];
}

behavior initializeSvc() {
//=====
// This logic is executed in 0 simulated time to initialize an instance of this
// operating system on each computer that specifies ADNosSvc in the CSE.ops file.
// The "operating_system_name" (second behavior parameter) corresponds to the
// name in column one of the CSE.ops file. This name may be used to differentiate
// between the initialization of differently named operating systems.
//=====

    osSetData("operatingSystemInstance",this);

    osSetData("svcState",1); // required for initialization process

//-----
// Startup Memory Pageout Manager (required by memory model)
// (use priority of 100 for compatibility with rel 2.1)
//-----
    Startup fOsMemMgr = MemoryPageoutManager( fMemoryStruct, fPageSize,
        fInstrPerPage ) Priority 100;
    processSetName(fOsMemMgr,"mpm-"+fComputerName);

// OS service Master controls affect all operating system instances that
// specify use of the ADNosSvc behavior in column 6 of the CSE.ops file.

// active tasks control process

```

```

fTaskCountMax = osGetData("taskCountMax");
If ( fTaskCountMax >= 0 ) {
    Call initTaskMgr();
}

// NFS server process
Startup fOsNFS = ADNOSNFS( this ) Priority 100 Options "NoStatsFlag";
processSetNameOnly(fOsNFS, "nfs-"+computerGetName());
registerSendDistributionPolicy2( fOsNFS, fRemoteIoDistributionPolicy );

// volume manager
Startup fOsVolMgr = ADNOSVolumeMgr() Priority 100;
}

behavior executeSvc() {
//=====
// this logic is executed in 0 simulated time to send any volume or remote IO requests
// included in an Execute statement to the local Volume manager or NFS server
//=====
variable i; // index variable
variable execSize; // number of request elements in the execute statement
variable reqType; // request element type: LocDisk, RemDisk, LocVol, RemVol, NonIo
variable ioReq; // I/O request handle
variable Kbytes; // Size in Kbytes of an I/O request
variable KbytesOffset; // Offset in file of first byte of data
variable volumeHandle; // Handle to volume where I/O data is located

execSize = hrGetData(HRexecSize);
i = 0;
While( i < execSize ) {
    reqType = hrGetDataX(HRexecReqType,i);
    Switch ( hrGetDataX(HRreqType,i) ) {
        Case( LocDiskType ) {
            // no OS service required
        }
        Case( RemDiskType ) {
            ioReq = hrGetDataX(HRioReq,i);
            Kbytes = hrGetDataX(HRkbytes,i);
            Send fOsNFS ("client_side",hrGetDataX(HRnfsProc,i),
                ioReq,reqType,Kbytes,0,0); // async
        }
    }
}

```

308 ↗

```

        hrSetDataX(HRresetReq,i,0);
    }
    Case( LocVolType ) {
        ioReq = hrGetDataX(HRioReq,i);
        Kbytes = hrGetDataX(HRkbytes,i);
        KbytesOffset = hrGetDataX(HRkbytesOffset,i);
        volumeHandle = hrGetDataX(HRvolumeHandle,i);
        Send fOsVolMgr (0,ioReq.reqType,Kbytes,KbytesOffset,
            volumeHandle,0); // async
        hrSetDataX(HRresetReq,i,0);
    }
    Case( RemVolType ) {
        ioReq = hrGetDataX(HRioReq,i);
        Kbytes = hrGetDataX(HRkbytes,i);
        KbytesOffset = hrGetDataX(HRkbytesOffset,i);
        volumeHandle = hrGetDataX(HRvolumeHandle,i);
        Send fOsNFS ("client_side",hrGetDataX(HRnfsProc,i),
            ioReq.reqType,Kbytes,
            KbytesOffset,volumeHandle); // async
        hrSetDataX(HRresetReq,i,0);
    }
    Case( NonIoType ) {
        // no OS service required
    }
    }
    i = i + 1;
}
return( EXECUTESvc );
}

//=====
// The logic in the following four behaviors: sendSvc, sendWaitSvc, receiveSvc, replySvc
// is invoked on all application state logic originating from send/wait receive/reply
// ADN statements
//=====
behavior sendSvc(aMsg) {
    variable tMsg;
    tMsg = ses_Message.associatedMsg( aMsg );
    // < Insert optional logic here >
    tMsg.sendToHardware(tMsg.receiving_proc_sn,tMsg.message_bytes);
}

```

FIGURE 3 PAGE - 10 -

```

        return( SENDsvc );
    }
    behavior sendWaitSvc(aMsg) {
        variable tMsg;
        tMsg = ses_Message.associatedMsg( aMsg );
        // < Insert optional logic here >
        return( SENDWAITsvc );
    }
    behavior receiveSvc(aMsg) {
        variable tMsg;
        tMsg = ses_Message.associatedMsg( aMsg );
        // < Insert optional logic here >
        return( RECEIVESvc );
    }
    behavior replySvc(aMsg) {
        variable tMsg;
        tMsg = ses_Message.associatedMsg( aMsg );
        tMsg.sendToHardware(tMsg.receiving_proc_sn,tMsg.message_bytes);
        // < Insert optional logic here >
        return( REPLYsvc );
    }

//=====
// Maximum task control management
// o Keeps track of all active threads executing on computing node
// o Is controlled via the corresponding entry in the CSE.ops file
//=====

// ----- logic for release 3.0

behavior taskStartSvc( thid ) {
    fActiveTaskCount = fActiveTaskCount + 1;
    fActiveTaskCountStats.sample(1.0);
    If ( fTaskMaxReached == false ) {
        If (fActiveTaskCount == fTaskCountMax) {
            if ( fTaskMaxWarningIssued == false ) {
                fTaskMaxWarningIssued = true;
                Warning "***** First maximum concurrent task count reached.\n",
                " Check trace file for time of first occurrence and computer name for each computer.\n",
                " Check report file \"Custom Statistics\" for active task count statistics for each computer.";
            }
        }
    }
}

```

```

    }
    fTaskMaxReached = true;
    Print stringFormat("%.6f", simGetTime()),
    "***** Maximum concurrent task count limit reached for computer",
    "\n"+fComputerName+"\n";
    }
    return( TASKSTARTsvc );
}

behavior taskEndSvc( thid ) {
    fActiveTaskCount = fActiveTaskCount - 1;
    fActiveTaskCountStats.sample(-1.0);
    return( TASKENDsvc );
}

behavior initTaskMgr() {
    // create active task count user stat
    tStatsName = ses_ComputerStatName(fComputerName);
    if ( associativeArrayElementsDefined(fActiveTaskCountStatsPtr, tStatsName) ) {
        fActiveTaskCountStats = fActiveTaskCountStatsPtr[tStatsName];
    }
    else {
        fActiveTaskCountStats = ses_gStatMgr.createContinuousStatistic("TaskMgr_activeTasks_" + tStatsName);
        fActiveTaskCountStatsPtr[tStatsName] = fActiveTaskCountStats;
    }
}

// -----
// NFS server behavior
// -----

Behavior ADNosNFS( aServer ) {
    variable tExecuteProc;
    real tMsgSendLength;
    real tMsgReplyLength;

    osSetData("svcState", 1);

```

FIGURE 3 PAGE - 12 -

```

processSetNoThreadUtilizationStats();

While( 1 ) {
    Receive( aType, arg1,arg2,arg3,arg4,arg5,arg6 ) Thread {
        Switch( aType ) {
            Case( "client_side" ) {
                // save client process id
                tExecuteProc = messageGetSendingProcessId();
                processSetClientId( tExecuteProc ); // c_proc->client_proc_sn = execute_proc
                If ( arg3 == ReadType ) {
                    tMsgSendLength = RPCreadReq;
                    tMsgReplyLength = arg4;
                }
                Else { // WriteType
                    tMsgSendLength = arg4;
                    tMsgReplyLength = RPCwriteAck;
                }
            }

            // forward request to remote server
            send arg1("server_side", arg2,arg3,arg4,arg5,arg6,tExecuteProc) Message tMsgSendLength
            Protocol "UDP/IP" Wait();

            // post completion event to Execute statement synchronization control
            // If ( ! arg6 ) { // not a volume manager request -- bug 3225 fix

            hrSetData(HRpostExecute,tExecuteProc);

            // }

            Case( "server_side" ) {
                processSetClientId( arg6 ); // execute_proc
                If ( arg5 ) {
                    // volume request
                    // tMsgReplyLength = 0.0;
                    tMsgReplyLength = arg3; // bug 3225 fix
                    Send ( aServer.fOs VolMgr) ( arg6,arg1,arg2,arg3,arg4,volGetLocalHandle(arg5),threadGetCurrentId()); //

                    threadWaitForSignal();
                }
                Else {

```

async

```

// disk request
If ( arg2 == ReadType ) {
    tMsgReplyLength = arg3;
}
Else { // WriteType
    tMsgReplyLength = RPCwriteAck;
}
// issue local IO request
hrSetData(HRlocalIO,arg1);
    }
    } Reply() Message tMsgReplyLength;
}

//-----
// Volume manager behavior
//-----

Behavior ADNosVolumeMgr() {

    // thread variables (separate copy for each)
    variable kbytes;
    variable kbytes_offset;
    variable request_start_time;
    variable stripe_kbytes;
    variable mod_kbytes_offset;
    variable curr_kbytes;

    osSetData("svcState",1);
    processSetNoThreadUtilizationStats();

    While( 1 ) {
        Receive( execute_proc, io_req, req_type, req_kbytes, first_kbytes_offset, volume_handle, waitId ) Thread (
            osSetData("svcState",1);

            If ( !execute_proc ) { // local request
                execute_proc = messageGetSendingProcessId();
            }
        }
    }
}

```

FIGURE 3 PAGE - 14 -

```

}
processSetClientProcessId( execute_proc );

// collect statistics
volBeginRequest( volume_handle, execute_proc );
request_start_time = simGetTime();

// for each volume IO request in Execute statement
kbytes_offset = first_kbytes_offset;
kbytes = req_kbytes; // total bytes in this I/O request ( Bytes * Number )
stripe_kbytes = volGetStripeSize(volume_handle);

// process first stripe, partial stripe up to a stripe boundary, or full request
mod_kbytes_offset = RMod(kbytes_offset,stripe_kbytes);
curr_kbytes = RMin(stripe_kbytes-mod_kbytes_offset,kbytes);

// modify Number field of original request (to avoid setting it each time)
hrSetDataX(HRReqNumber,io_req,1);

// save the disk number as the reference point for a complete pass through
// all of the disks in the volume
first_disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);

Join {

    // loop until all of the data has been processed

    While ( kbytes > 0.0005 ) {

        Join {

            disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);

            // loop over each disk on volume once while there is more data

            While ( ( disk_number >= 0 ) && ( kbytes > 0.0005 ) ) {

                // for each piece of an I/O request
                Thread {

                    // declare client process for associating statistics

```

FIGURE 3 PAGE - 15 -

```

processSetClientProcessId( execute_proc );

If ( kbytes > curr_kbytes ) {
    // copy original I/O request
    ioReq = hrGetDataX(HRioReqCopy,io_req);
}
Else {
    // use original I/O request
    ioReq = io_req;
}

// modify selected fields
hrSetDataX(HRioReqDisk,ioReq,disk_number);
hrSetDataX(HRioReqKbytes,ioReq,curr_kbytes);

// issue local IO request
hrSetData(HRlocalVIO,ioReq);
}
kbytes_offset = kbytes_offset + curr_kbytes;
kbytes = kbytes - curr_kbytes;
curr_kbytes = RMin(stripe_kbytes,kbytes);

disk_number = volGetStripedDiskNumber(volume_handle,kbytes_offset);
If ( disk_number == first_disk_number ) {
    disk_number = -1;
}
} // While - loop over each disk on the volume once while there is more data

} // Join - wait here until all the disks have completed

} // While - loop while there is more data to be processed

} // Join - wait here until all the data has been processed and all of the threads completed

// post completion event to Execute statement synchronization control
// when all pieces of this request have been completed

if ( waitId ) {
    // request from NFS
    threadSignal(waitId);
}

```

FIGURE 3 PAGE - 16 -

```

    }
    else {
        // local request
        hrSetData(HRpostExecute, execute_proc);
    }

    // collect statistics
    volEndRequest( volume_handle, execute_proc, request_start_time );

    } Reply();
}

// =====
// -----
// this is a required operating system factory behavior
// its name should appear in column 6 of the CSE.ops file for all
// named operating systems that use the OperatingSystem class
// -----

public behavior ADNosSvc( aSvcType, aArg2, aComputerName,
    aMemoryStructure, aPageSize, aInstrPerPage ) {
    variable t_OpSys;
    if ( aSvcType == 0 ) {
        tOpSys = new ses_OperatingSystem( aArg2, aComputerName,
            aMemoryStructure, aPageSize, aInstrPerPage );
        call tOpSys.initializeSvc();
    }
}

```

FIGURE 3 PAGE - 17 -